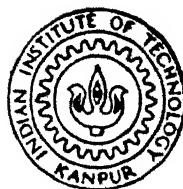


# ALGORITHMIC ASPECTS OF NATURAL LANGUAGE PARSING USING PANINIAN FRAMEWORK

*By*

**Bendapudi Perraju V. S.**



Department of Computer Science and Engineering  
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

DECEMBER, 1992

CSE  
1992  
M  
BEN  
ALG

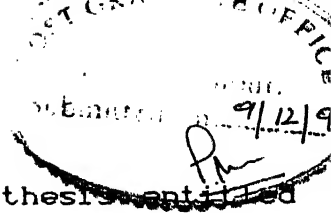
# **ALGORITHMIC ASPECTS OF NATURAL LANGUAGE PARSING USING PANINIAN FRAMEWORK**

*A Thesis Submitted  
in Partial Fulfilment of the Requirements  
for the Degree of*  
**MASTER OF TECHNOLOGY**

*By*  
**Bendapudi Perraju V. S.**

*to the*  
**Department of Computer Science and Engineering**  
**INDIAN INSTITUTE OF TECHNOLOGY KANPUR**  
**DECEMBER, 1992**

CERTIFICATE

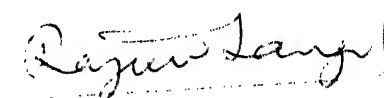


It is certified that the work contained in this thesis entitled  
ALGORITHMIC ASPECTS OF NATURAL LANGUAGE PARSING USING PANINIAN  
FRAME WORK by B.V.S.Perraju has been carried out under our  
supervision and that this work has not been submitted elsewhere  
for a degree.

  
(Dr. Somnath Biswas)

Professor

Dept. of Computer Sc. & Engg.

  
(Dr. Rajeev Sangal)

professor

Dept. of Computer Sc. & Engg.

IIT Kanpur

December 1992

DEDICATED TO MY PARENTS

**24 FEB 1993**

**CENTRAL LIBRARY**

**Acc. No. A.114837**

## ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to Dr. R. Sangal and Dr. S. Biswas for their invaluable guidance and constant encouragement throughout the thesis work. I am indebted to Dr. R. K. Ahuja and Dr. V. Chaitanya for their timely advice and help during crucial moments.

My very special thanks to Ravi, Siva, Sokkalingam, Das, Venu and Reddy without whose help this thesis would never have reached this shape.

I am thankful to all my friends Murali(my guru), Sridhar, OSM, Suresh, Srikanth, TSR, Malles, Sharma, Ramudu, Karra, Venku, Vissu, Subba, Potti, Kalari, AP, DEV, Pilla, Veera, Bhairu, Narayana & B. Srinivas, who made my stay at IIT Kanpur a memorable one.

## ABSTRACT

The MT system at IIT Kanpur uses paninian framework. It consists of a morphological analyzer, local word grouper and a core parser collectively referred to as the parser. The core parser uses karaka charts which specify the syntactico-semantic relations, called the karaka relations, between verb and noun groups (also known as demand groups and source groups respectively). The parsing problem should satisfy certain general constraints in addition to the above specified constraints. This thesis will describe the complexity aspects of the system with these constraints.

Two approaches are taken to get the complexity of the parsing problem. In the first approach the parsing problem was reduced to already well known problems. There are two types of karakas, mandatory and optional. With some constraints the problem was reduced to bipartite matching problem and With the constraint which ensures that all mandatory karakas must be filled the problem was reduced to maximum matching. Finally with some additional constraints the problem was reduced to Min cost flow problem.

In the second approach the parsing problem was formulated as integer programming problem. With some of the constraints the matrix of the turns out to be unimodular and totally unimodular.

## TABLE OF CONTENTS

1.	Introduction	1
1.1	Paradigms of MT	2
1.2	IIT-Kanpur MT system	3
1.3	Constraints and Complexity aspects of Parser	3
1.4	Outline of the thesis	5
2.	Paninian framework for NLP & Constraint formulation	6
2.1	Overview of the MT system	6
2.1.1	Morphological analyzer	6
2.1.2	Local word grouper	7
2.1.3	Core Parser	9
2.1.3.1	Karaka charts	10
2.1.3.1.1	Optionality of karakas	11
2.1.3.1.2	Karaka-vibhakti mapping	11
2.1.3.1.3	Semantic type	11
2.1.3.2	Karaka relations among word groups	13
2.1.4	Constraints	15
2.2	Constraint Parser	16
3.	Algorithmic aspects of the core parser	19
3.1	With constraints 1, 2 and 3	19
3.1.1	Reduction to Bipartite Graph	19

3.1.1.1	Formation of U	20
3.1.1.2	Formation of V	20
3.1.1.3	Formation of Edges	21
3.1.2	Bipartite Matching Algorithm	23
3.2	Introduction of Constraint 4	24
3.2.1	Assigning 0,1 weights to edges	25
3.2.2	Maximum bipartite matching	25
3.3	Introduction of Constraint 5	27
3.3.1	Min cost flow problem	28
3.3.2	Formation of G	28
3.3.2.1	Formation of VV	28
3.3.2.2	Formation of E	29
3.4	Integer programming approach	32
3.4.1	With Constraints 1 to 5	34
3.4.2	With Constraints 1 to 6	34
3.5	Complexity aspects	42
3.5.1	Increasing no. of merged karaka charts	43
3.5.2	Allowing source word, demand word clashes	44
4.	Conclusions	45
References		

## LIST OF FIGURES

2.1	Flowchart of the IIT-Kanpur MT system	6
2.2	Semantic type hierarchy	12
2.3	Karaka chart of KACeat)	12
2.4	Constraint graph of example 2 sentence	14
2.5	Flow chart of Core parser	18
3.1	Filter flowchart	22
3.2	Flowchart of the core parser with constraints 1 to 3	24
3.3	Flowchart of the core parser with constraints 1 to 4	26
3.4	Graph $G=(V,V,A)$ for example 2	31
3.5	Flowchart of the core parser with constraints 1 to 5	31
3.6	Graph of example 3	35
3.7	Graph of example 6	40
3.8	Reduced graph of example 6	41
3.9	Flowchart of the core parser	42

# CHAPTER 1

## INTRODUCTION

Translation is the process of converting text in one language to another language such that its meaning is preserved. If that process is done using a computer, it is called **Machine Translation** (referred to as **MT**). Translation is difficult because each language has its idiosyncrasies like

1. words are overloaded to represent multiple concepts
2. sentence constructions may vary in the source and target languages.

So to capture the essence of a sentence the use of a proper intermediate representation is essential.

The Machine Translation problem is dependent on the choice of the internal representation and strategies for the internal representation construction from source text. The MT problem after the choice of internal representation can be divided into two parts.

1. Parsing and
2. Generation.

Parsing is the process of assigning a suitable structure called the parse structure that captures the internal relationships between words in the given source sentence.

Generation is the process of constructing the target language sentence given the parse structure.

## 1.1 Paradigms of MT:

There are three different strategies based on the amount of information present in the parse structure and the manner in which the generator uses it. These strategies are used by different MT systems.

**Direct MT strategy:** In this strategy, no general linguistic theory or parsing principles are necessarily present. The system relies on well developed dictionaries, morphological analysis and text processing software to gain credible translations of the source language text into a series of reasonably equivalent words and phrases in the target language. The Georgetown system, to translate English to Russian, adopts this strategy.

**Transfer MT strategy:** In this strategy, a source language text is parsed into an abstract internal representation. A transfer is made at both the lexical and structural levels to the target language and the translation is generated. For this, source language, target language, bilingual lexicons are required. Source language lexicon is used in parsing. The levels at which the transfer occurs, differs from system to system, ranging from syntactic deep structure markers to syntactico-semantic representations.

**Inter-lingua MT strategy:** In this strategy, the source text is parsed and is mapped to a languagefree conceptual representation. The language used to represent the conceptual information is known as the interlingua. Inference mechanisms then apply contextual and world knowledge to augment the representation. Finally the

generator maps the appropriate sections of the languagefree representation to target language.

## **1.2 IIT-Kanpur MT system :**

It uses the third strategy specified above, i.e., interlingua approach. The parse structure encapsulates as much conceptual information as required for the purpose of translation. The parse structure is centered around the verbal groups of the sentence and is based on the karaka relations. The karaka relations are syntactico-semantic relations between the verbal groups and the source groups in the sentence. The parse structure is a mapping between the source groups and the karaka relations. Besides the karaka relations the sentence may contain the non-karaka relations such as those contributed by the adjectival relations, purpose and relational words, etc. The karaka relations are used for the disambiguation of word senses. Thus the parse structure contains a mapping between the source groups and the karaka relations along with the concepts of the various words in the text.

## **1.3 Constraints and Complexity aspects of Parser:**

As mentioned earlier the parser is a main part in the machine translation. So the time taken to translate a source language sentence to a target language sentence is dependent on the time taken to form a parsed structure.

The MT system uses a constraint parser, which will use constraints to get suitable candidates for karaka roles of verb groups. To get a unique parse structure, in addition to the gnp (gender,number,person) filter, karaka-vibhakti filter and semantic filter (feature constraints), six additional constraints are introduced (which are applicable to natural languages). These constraints will ensure, to a certain degree, the unique parse structure.

The core parser was implemented using integer programming. The constraints are formulated as integer programming equations. But the parsing problem complexity is not known, because the problem complexity may not be equal to the algorithm' complexity. It may be less than or more than the algorithm complexity. This thesis will describe the complexity aspects of the parsing problem.

Two approaches are used to study the complexity aspects of the parsing problem.

In the first one, the parsing problem with some of the constraints was reduced to the other known problems, like matching problem, mincost flow problem. In the second approach the problem with all the six constraints was formulated as an integer programming problem and the total unimodularity and unimodularity conditions were tested on the coefficient matrix of the integer programming formulation.

The complexity aspects of the system with category clashes and K merged karaka charts for each verb group are also

discussed.

#### 1.4 outline of the Thesis:

The second chapter describes the details of the MT system and the constraints. The third chapter describes the complexity aspects of the system. Conclusions and pointers for further enhancements follow in the last chapter.

## CHAPTER 2

### PANINIAN FRAMEWORK FOR NLP & CONSTRAINT FORMULATION

#### 2.1 Overview of the MT system:

The flow chart of the MT system is shown in Fig 2.1

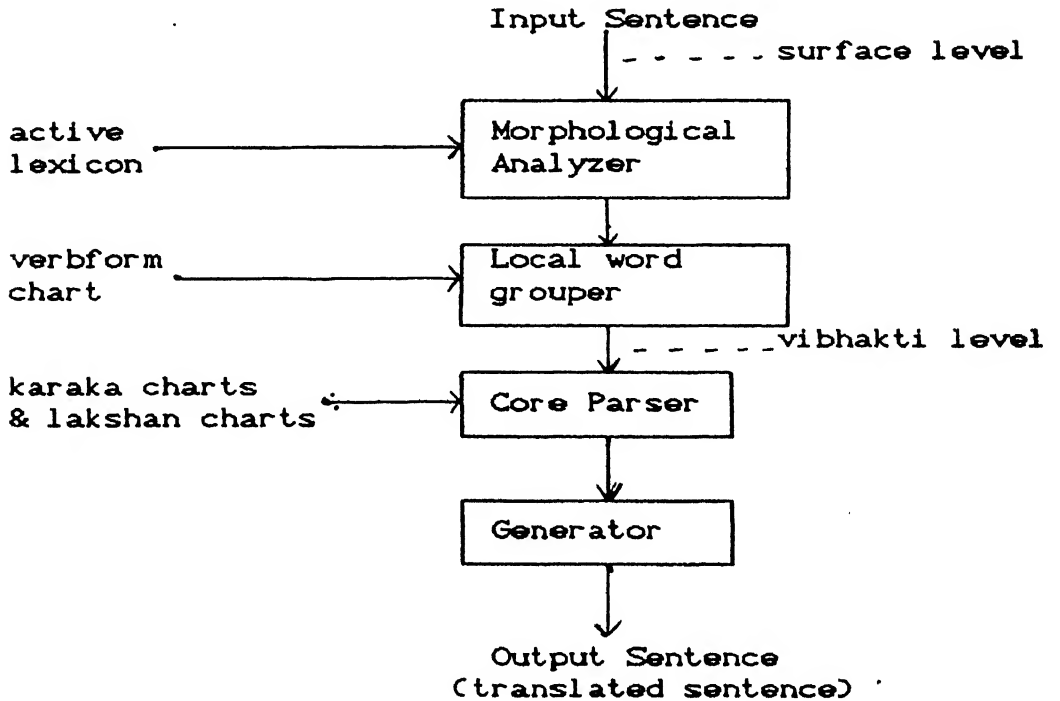


Fig 2.1 Flowchart of the IIT-Kanpur MT system

Each box is described in detail below.

##### 2.1.1 Morphological analyzer:

At this stage the grammatical information of each word in a given sentence is obtained. For each word the category it belongs to and the feature values associated with the category are obtained.

The categories are language dependant. For Hindi, different categories are Noun, Verb, Adjective etc. Different features associated with categories are

1. Gender, Number, Person for Nouns

2. Gender, Number, Person and part of TAM (Tense, Aspect and Modality) for Verbs.

(In case of Telugu Verbs, complete TAM label is obtained.)

The Morphological analyzer retrieves, for each word, to which category the word belongs, What are the Gender, Number and Person values if the word is a Noun, what are the Gender, Number, Person and TAM values if the word is a verb.

If the word belongs to more than one category (category clash) or if the word has more than one set of feature values then the morphological analyzer will return information for each of them.

### 2.1.2 Local Word Grouper:

Though Indian languages are relatively word order free, some units follow word order like main verb is followed by auxiliary verbs, Nouns are followed by Vibhakti's etc. Such units combine into groups based on local information in the local word grouping stage. Vibhakti is the word combined with a Noun in the Noun group. A noun group may contain a 0-vibhakti (null string as vibhakti, i.e., No vibhakti)

example 1:

Ram Pala ko KAtA hE

(Ram eats the fruit)

In the above sentence 'RAM', 'Pala', 'ko', 'KAtA', 'hE' are given words. Morphological analyzer retrieves information for each of them. Here 'Ram', 'Pala' are Nouns, 'KAtA' is main verb, 'hE' is auxiliary verb, 'ko' is vibhakti. The local word grouper will form word groups from these words. The noun 'Pala' and vibhakti 'ko' are combined to form a noun group. The main verb 'KAtA' and the auxiliary verb 'hE' are combined to form a verb group. Such grouping is done at the local word grouping stage. In the example 1 'Ram' is a noun group with 0-vibhakti.

In the local word grouping, the verb grouping is done based on possible verb sequences that may occur and information about agreement, the noun groups are composed on the form of the noun and the following vibhakti (post position marker). However, in those cases where there is ambiguity in identifying the local word groups and the ambiguity cannot be resolved at that stage, the decision is postponed. For example, In Hindi, a word that can be both a noun and an adjective causes ambiguity in forming a local word group with its succeeding noun.

The noun groups and verb groups are also known as source groups and demand groups.

The phenomenon of local word grouping occurs to a greater extent in languages like Hindi where as the declension takes the form of a postposition and the auxiliary verbs are

separated by word boundaries. In languages such as Telugu, local word grouping occurs to a lesser extent, since a word on most occasions, is morphologically inflected.

Let  $x_1, x_2, \dots, x_p$  be the given input sentence, where each  $x_i$  is a word group.

If  $w_1, w_2, \dots, w_n$  are verb groups

and  $s_1, s_2, \dots, s_m$  are noun groups

(where  $m+n \geq p$ )

and  $(L_i, w_j)$  is the lexical information of the  $i$ th meaning of word group  $j$  ( $w_j$ ) then the local word grouper will return

$\{ (L_1 w_1, L_2 w_1, \dots), (L_1 w_2, L_2 w_2, \dots), \dots, \dots, (L_1 w_n, L_2 w_n, \dots) \}$

and  $\{ (L_1 s_1, L_2 s_1, \dots), (L_1 s_2, L_2 s_2, \dots), \dots, \dots, (L_1 s_m, L_2 s_m, \dots) \}$

### 2.1.3 Core Parser:

If a word group belongs to both source group and demand group, local word grouper will give both informations. If category clashes (a word group belongs to both source group and demand group) are there, before using the local word grouper output for the parser algorithm, only one set of information is taken with each word. If parse structure is not generated then the parser algorithm has to execute again with a different set of information.

The complexity of the system increases if every word group belongs to every category. If  $K$  clashes ( $K$  word groups having category clashes) are there then the constraint solver (core parser) has to be executed  $2^K$  times (number of groups taken as 2) in the worst case. The complexity aspects are discussed in chapter 3.

The task of the core parser is to identify karaka relations among word groups. It requires the karaka charts. There is a separate karaka chart for each verb group in the sentence being processed.

#### 2.1.3.1 Karaka charts:

The karaka chart of a verb group will store information about the different karaka's associated with it. They are also termed as karaka roles of the verb groups. The noun groups are to be assigned to these karaka roles of the verb groups. The verb groups are called the demand groups as they make demands about their karakas, and the noun groups are called source groups because they satisfy such demands. (A verb group can be a source group as well when it satisfies the demand of another verb group. This however, does not affect its status as a demand group as well.

With each karaka, karaka restrictions are also stored in the karaka chart. There are 3 different restrictions.

1. Optionality of karakas
2. Karaka-vibhakti mapping
3. Semantic type information

#### 2.1.3.1.1 Optionality of karakas:

For each karaka role this field will specify whether that karaka is Mandatory or Optional. If it is mandatory then there must be an assignment for that karaka role, i.e., some source group must be assigned to that karaka. If it is optional then the karaka may or may not be filled. (this constraint is discussed in section 2.1.4.)

#### 2.1.3.1.2 Karaka-vibhakti mapping:

Each source group (Noun group) will have a vibhakti (can be a 0-vibhakti) with it. Each karaka will specify some vibhakties (against each karaka the acceptable vibhakties are given in the karaka chart. example karaka chart is shown in Fig 2.3). Source groups having those vibhakties must be assigned to that karaka role. This constraint is also termed as Feature constraint, because it is restricting the feature aspect of source group.

#### 2.1.3.1.3 Semantic type:

This information is limited to that necessary for removing ambiguity, if any, in karaka assignment. In other words, for a given verb, when karaka-vibhakti mapping is not sufficient for producing a parse structure, semantic types are included. The semantic types included have the sole purpose of karaka disambiguation. This keeps the number of semantic types under control, and serves as a guiding philosophy for what semantic

types to include. The Fig 2.2 shows a possible semantic type hierarchy which is sufficient for a major part of language.

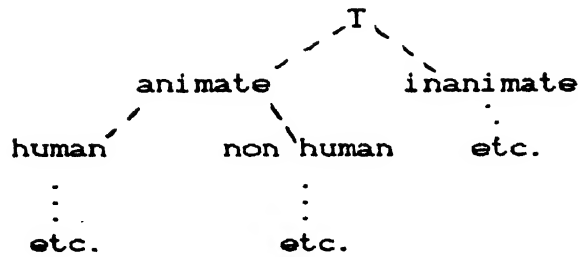


Fig 2.2 Semantic type hierarchy

It cannot be a tree always. Some times it will be a graph.

An example karaka chart for KA (eat) is given in Fig2.3.

karaka	optionality	vibhakti	semantic type
karta	m	0	animate
karma	m	0 or ke	--
karana	o	se	--

Fig 2.3 karaka chart of KA (eat)

The gender, number and person agreement between verb groups and noun groups is verified in the GNP filter of the core parser ( language grammar rules are used for the agreement).

A verb group can have more than one karaka chart. In that case with one selection the core parser has to be executed. If parse structure is not generated then with another karaka chart

selection the core parser algorithm has to be executed. If there are  $n$  verb groups and each verb group  $V_i$  is having  $kk_i$  ( $i=1$  to  $n$ ) karaka charts then the core parser algorithm has to be executed  $k_1 * k_2 * \dots * k_n$  times in the worst case. For that reason all the karaka charts of a verb group are merged. The merging procedure is given below.

The union of karakas of all karaka charts of the verb group will be taken in the merged karaka chart. The karaka's which are mandatory in all karaka charts of that verb group will remain mandatory in the merged karaka chart. All remaining karakas will be optional in the merged karaka chart. For the remaining fields, the union of all field values will be taken.

The merged karaka chart so formed is used in the core parser algorithm. The other aspects of merged karaka chart are discussed in chapter 3.

#### 2.1.3.2 Karaka Relations among word groups:

For a given sentence after the word groups have formed, karaka charts for the verb groups are identified (first two phases) and each of the noun groups is tested against the karaka restrictions in each karaka chart (provided the noun group is to the left of the verb group is to the left of the verb group whose karaka chart is being tested.). When testing a noun group against a karaka restriction of a verb group, vibhakti information and semantic type are checked and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group. This

can be shown in the form of a constraint graph. Nodes of the graph are the word groups and there is an arc from a verb group to a noun group labelled by a karaka, if the noun group satisfies the karaka restriction in the karaka chart of the verb group (There is an arc from one verb group to another verb group, if the karaka chart of the former has a karaka restriction with semantic type as action). The verb groups are called demand groups as they were demands about their karakas, and the noun groups are called source groups because they satisfy demands (A verb group can be a source group as well when it satisfies the demand of another verb group).

As an example, consider a sentence containing the verb KA (eat) with its word groups marked.

example 2: baccA kele ko KAtA hE

child banana -ko eats

(The child eats the banana)

Its constraint graph is shown in Fig 2.4.

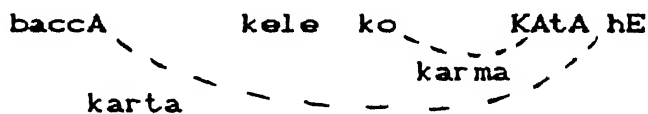


Fig 2.4 Constraint graph of example 2 sentence.

It also happens to be the solution graph because all source groups are assigned.

Consider another sentence where the constraint graph is different from the solution graph.

baccA kela KAtA hE

(child eats banana)

Here both names qualify to be karta and karma (in the previous example 2 vibhakti 'ko' in source group 'kele ko' qualified it to karma.). In such a situation, the parser produces both parses. To get a unique parse structure some additional constraints are applied.

For example, if we put karta as animate then the parse structure where 'baccA' as karta and 'kelA' as karma will be selected.

Some additional constraints are added to get unique parse structure. In terms of the constraint graph a parse is a sub-graph of the constraint graph satisfying some constraints. The constraints are given in section 2.1.4.

#### 2.1.4 Constraints:

1. From a demand node, for all mandatory karaka labels, out of all edges with a particular mandatory karaka label, only one edge has to be selected.

(A mandatory karaka of a verb group must be filled with one source group only.)

2. From a demand node, for all optional karaka labels, out of all edges with a particular optional karaka label, no edge or only one edge has to be selected.

(A optional karaka of a verb group must be filled at most with one source group only.)

3. A source node can be assigned to only one demand node, for all labels, for all demand nodes.

(A source group must be assigned to only one karaka role.)

4. For all demand nodes, the edges labelled by mandatory karakas must be selected first.

(All mandatory karakas must be filled.)

5. From a demand node, out of all outgoing edges with a set of  $m$  optional karaka labels,  $n$  edges must be selected where  $n \leq m$

(Out of  $m$  optional karakas,  $n$  optional karakas must be filled.)

6. In the solution graph there shouldn't be any arc intersections.(nesting constraint.)

## 2.2 Constraint Parser:

Currently a parse is obtained from the constraint graph using integer programming. The constraint graph, with additional constraints specified in section 2.1.4 is converted into an integer programming problem by introducing a variable for a edge from node  $i$  to  $j$  labelled by karaka ' $k$ ' in the constraint graph such that for every edge there is a variable. The variables take their values as 0 or 1. A parse is an assignment of 1 to those variables whose corresponding edges are in the parse sub-graph, and 0 to those that are not. Equality and inequality constraints in integer programming problem can be obtained from the

constraints listed in section 2.1.4. For some of the constraints the formation of equality and inequality constraints is given below.

1. For each demand group  $i$ , for each of its mandatory karakas  $k$ , the following equality constraints  $M$  must hold.

$$M(i,k) : \quad \sum_j x_{ij,k} = 1.$$

Thus there will be equality constraints  $M(i,k)$  corresponding to mandatory karakas for each of the demand words.

2. For each demand group  $i$ , for each of its optional karakas  $k$ , the following inequalities must hold

$$O(i,k) : \quad \sum_j x_{ij,k} \leq 1.$$

Thus there will be equality constraints  $O(i,k)$  corresponding to optional karakas for each of the demand words.

3. Each source group  $j$  must be assigned only once

$$S(j) : \quad \sum_{ik} x_{ijk} = 1.$$

Thus there will be as many equality constraints  $S(j)$  as the source groups.

For the other conditions also the equalities are formulated.

The core parser flow chart is shown in Fig 2.5.

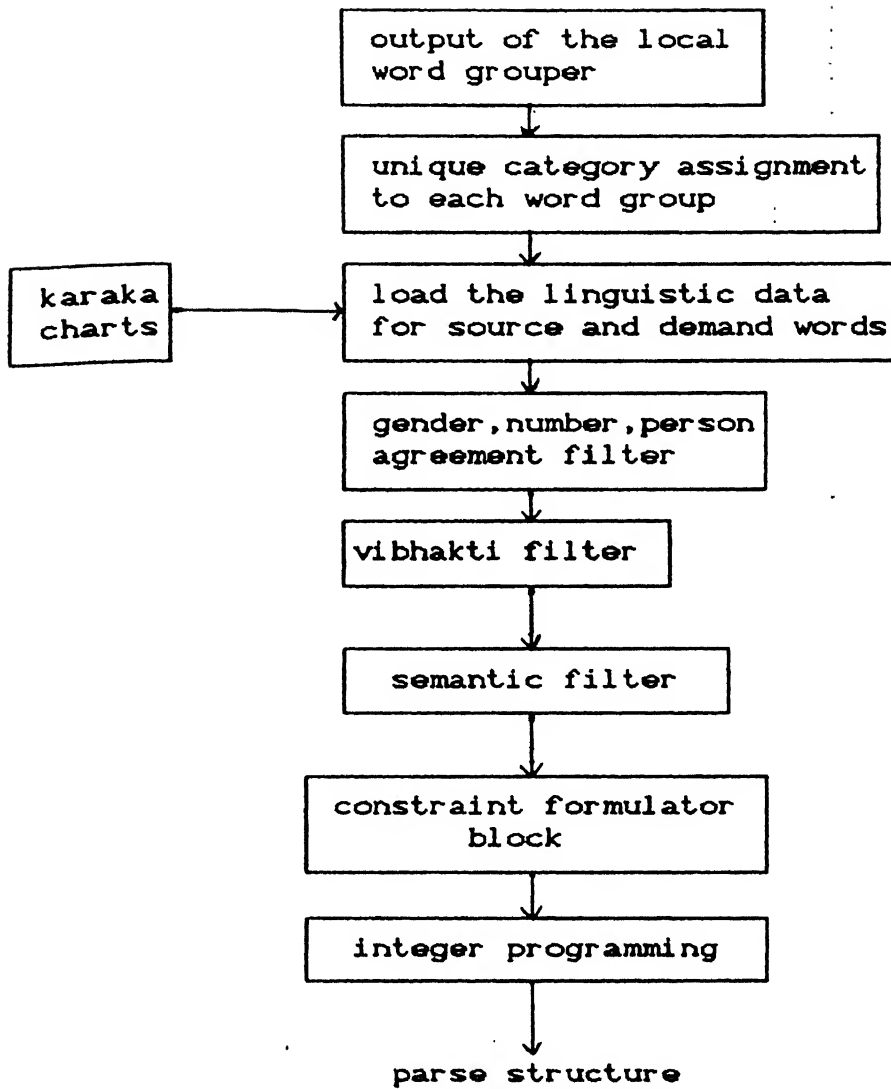


Fig 2.5 Flow chart of Core Parser.

The complexity of the parser with the constraints is discussed in the next chapter.

## CHAPTER 3

### ALGORITHMIC ASPECTS OF THE CORE PARSER

The core parser flowchart uses integer programming for which no polynomial time algorithm is known till now. We study here if this particular use of integer programming can be reduced to known polynomial time problems e.g. Assignment problem, Min-cost flow problem or Linear programming problem. For that study the constraints specified in section 2.1.4 are introduced in a regular fashion and the results are given.

#### 3.1 With Constraints 1, 2 and 3:

For constraints 1, 2 and 3 (which are given in 2.1.4), information from local word groups and karaka charts are captured in a bipartite graph and the parsing problem is solved by bipartite matching algorithm.

##### 3.1.1. Reduction to Bipartite Graph:

A bipartite graph  $G(U, V, E)$  is defined as

$U$  : set of nodes  $\{u_1, u_2, \dots, u_n\}$

$V$  : set of nodes  $\{v_1, v_2, \dots, v_m\}$

$E$  : edges between  $U$  and  $V$

and the constraint is  $U \cap V = \phi$ .

The reduction is done in three stages.

1. Formation of  $U$  from lexical information of source word Groups.

2. Formation of V from karakas of verb groups.
3. Formation of E from karaka relations among word groups.

Each one of the three stages are described in detail.

### 3.1.1.1 Formation of U:

Let the local word grouper produce

$$(L_1s_1, L_2s_1, \dots) (L_1s_2, L_2s_2, \dots) \dots (L_1s_m, \dots)$$

where  $s_1, s_2, \dots, s_m$  are source groups,

$L_1, L_2, \dots$  are lexical meanings,

and  $L_1s_j$  is the  $i$ th lexical meaning of source word  $s_j$ .

The set  $U = (u_1, u_2, \dots, u_m)$  is formed

where  $u_1 = (L_1s_1, L_2s_1, \dots)$

$u_2 = (L_1s_2, L_2s_2, \dots)$

...

..

$u_m = (L_1s_m, L_2s_m, \dots)$

### 3.1.1.2 Formation of V:

Let  $(w_1, w_2, \dots, w_n)$  be the verb groups in the sentence.

The karaka charts contain karaka information of these verb groups (i.e. which karakas are there with a verb group).

Using the karaka charts the output is

$$(k_1w_1, k_2w_1, \dots, k_tw_1) (k_1w_2, k_2w_2, \dots) \dots (k_1w_n, k_2w_n, \dots)$$

where  $k_1, k_2, \dots$  are acceptable karakas to the verb groups

and  $k_1w_j$  is the  $i$ th karaka role of the verb group  $w_j$ .

The demand set  $V = \{v_1, v_2, \dots, v_p\}$  is formed :

where  $v_1 = k_1 w_1$  :

$$v_2 = k_2 w_1$$

...

$$v_t = k_t w_1$$

..

$$v_i = k_i w_j$$

...

$$v_p = k_{(\text{last})} w_n$$

### 3.1.1.3 Formation of Edges:

Edges (E) are formed indicating possible karaka relations. An edge from  $u_i$  to  $v_j$  is formed if the source word group  $u_i$  is a candidate for the demand word  $v_j$ . Candidacy is determined by looking at the vibhakti constraint specified by the karaka charts, etc.

For each demand  $v_j$  acceptable candidates are selected applying the following filters.

Initially all source words to the left of the verb group (in the sentence) of the karaka are candidates. The filters given in Fig 3.1. are applied one by one to reduce the set of candidates acceptable. The filters can also be termed as feature constraints.

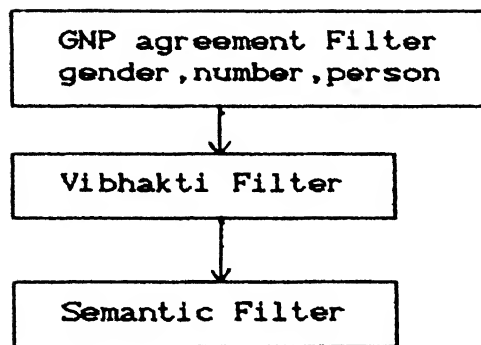


Fig 3.1 Filter Flow chart

For each  $v_j$  the filters are applied on the set  $\{u_1, u_2, \dots, u_x\}$ . (where  $\{u_1, u_2, \dots, u_x\}$  are source words to the left of verb group  $v_j$ )

If any  $u_i$  is an acceptable candidate then the edge  $u_i v_j$  is added to the edge set  $\{E\}$ . But here  $u_i$  is not single entry (some times). It can be a set. So the modified version of formation of edge set is

For each  $v_j$

if any of the  $L_k s_i$  is an acceptable candidate then

edge  $u_i v_j$  is added to the edge set  $\{E\}$

and label 'kj' is added to the edge.

In this new version extra information (label) is given with each edge.

If more than one  $L_k$  of the  $s_i$  are acceptable then the edge  $u_i v_j$  is added once and  $k$  is arbitrarily selected out of the satisfying  $L_k$ 's and  $kj$  is added.

Example 1:

Let  $v_j$  be the karaka we are dealing with.

Let  $u_1, \dots, u_a$  are the acceptable candidates initially

(source word groups to the left of the verb group  $v_j$ )

and  $(L_1s_1, L_2s_1, \dots)(L_1s_2, \dots) \dots (L_1s_a, \dots)$  are the lexical entries of source word groups.

If any of  $(L_1s_1, L_2s_1, \dots, L_bs_1)$  is an acceptable candidate for  $v_j$  then an edge  $(u_1, v_j)$  is added to the set  $(E)$  and label  $k_1$  is added to the edge (where  $L_k$  is the particular lexical entry of  $u_1$ ).

The formation of set  $U$  from the lexical information of source word groups can be done in linear time because its just an assignment of  $u_i$  to each source word group.

The formation of set  $V$  is linear time because it is also assigning  $v_j$  to karaka roles of verb groups.

The formation of Edges is of order  $O(|U| \cdot |V|)$ . So the formation of bipartite graph  $G = (U, V, E)$  can be done in polynomial time.

Now the bipartite graph is formed. This graph is given as input to the bipartite matching algorithm.

### 3.1.2 Bipartite Matching Algorithm:

A matching  $M$  of bipartite graph  $G = (U, V, E)$  is a subset of edges with property that no two edges of  $M$  share the same node. The matching problem is to find a maximum matching of  $G$ . An augmenting path algorithm is given in [papa82].

The algorithm will return a maximum cardinality matching, i.e., the cardinality of  $M = \min \{ |U|, |V| \}$ , if available.

In the output of the algorithm, the edges in matching are like  $u_i v_j$ . The label on that edge is taken and the proper  $L_k$  is found, then it is substituted instead of  $u_i$  and  $L_k v_j$  is the matching.

The flow chart of the core parser with constraints 1, 2 and 3 is given in Fig 3.2.

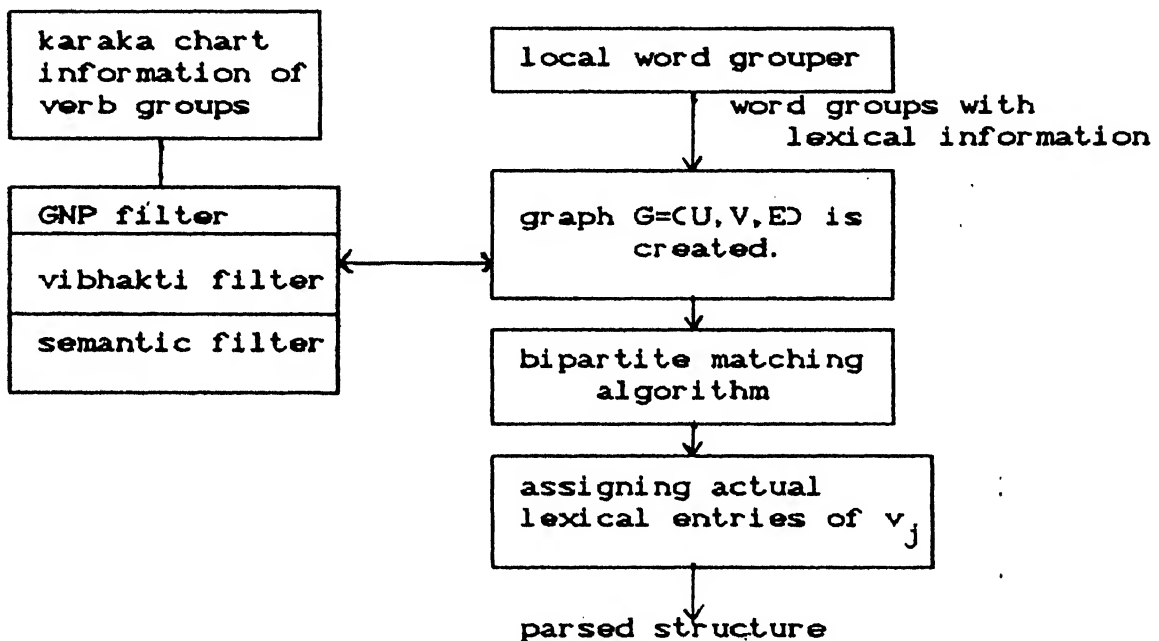


Fig 3.2. Flow chart of the core parser with constraints 1,2 and 3

### 3.2. Introduction of Constraint 4:

If distinction has to be made between mandatory and optional karakas(see constraint 4 in section 2.1.4) the bipartite matching algorithm is not sufficient. For all edges weights 0 or 1

are assigned and the bipartite matching algorithm is extended to maximum bipartite matching.

### 3.2.1 Assigning 0,1 weights to edges:

After the graph  $G$  is created, for each edge  $[u_i, v_j] \in E$ ,  $v_j$  is checked whether mandatory or optional, and weights are added to the edge.

If  $v_j$  is mandatory then weight 1 is assigned to the edge  $[u_i, v_j]$ .

If  $v_j$  is optional then weight 0 is assigned.

### 3.2.2. Maximum Bipartite Matching:

Given a graph  $G = (U, V, E)$  a number  $w_{ij}$  greater than or equal to 0 for each edge  $[u_i, v_j] \in E$  is to find a matching of  $G$  with the largest possible sum of weights. This ensures that all the mandatory karakas are covered before optional ones are considered.

After assigning weights (0,1) the graph  $G$  is given as input to the maximum bipartite matching algorithm [papa82] which will give a matching ensuring the constraint 4.

The assignment of weights is a polynomial time algorithm. Its complexity is  $O(E)$ .

The flow chart of the core parser is given in Fig 3.3.

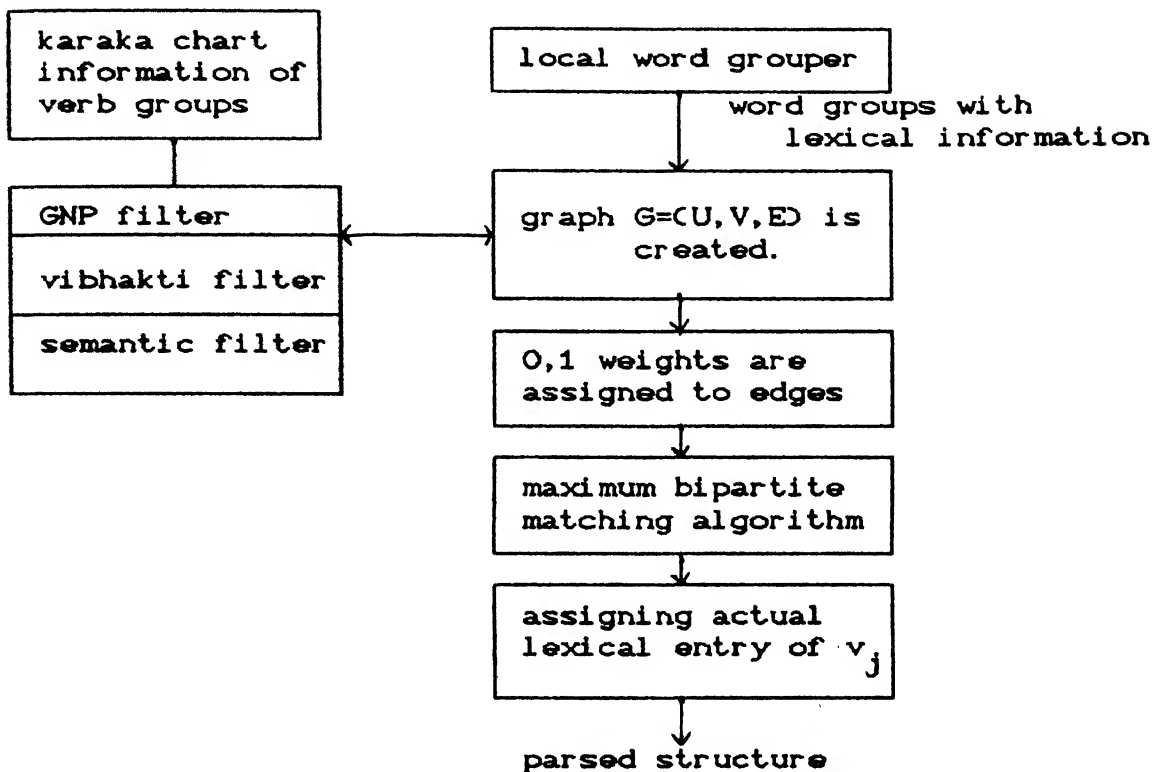


Fig 3.3. Flow chart of the core parser with Constraints 1 to 4

In the previous weight assignment method only 0 and 1 are assigned. They will ensure constraint 4. If more general weight assignment is needed (e.g., to specify source word  $s_1$  has more weight than source word  $s_2$  to become a candidate of  $v_j$ ) then the weight assignment block is modified.

If arbitrary weight assignment is done then the constraint 4 may not hold, because sometimes the weight of optional edges become higher than mandatory edges. So to ensure the constraint 4, weight assignment has to be done carefully. The procedure is

For all  $[u_i, v_j] \in E$

if  $v_j$  is optional then weight  $> 0$  is assigned to that edge, i.e., all edges having optional karakas are assigned weights first.

Let  $(x_1, x_2, \dots, x_y)$  are the optional karakas

Let  $M_i$  is the maximum assignment to any edge having  $x_i$ ,

then

$$MM = \sum_{i=1}^y M_i$$

For all edges having mandatory items the weight assigned to them must be  $> MM$

If assignment of weights is done by the above procedure then constraint 4 is ensured, by that all mandatory karakas will be filled. This weight assignment module replaces 0,1 weight assignment module in the flow chart of core parser. This block is of order of complexity  $O(|E|)$ . So max. weight matching will remain optimized matching.

### 3.3. Introduction of Constraint 5:

With the constraints 1,2,3 and 4 the parsing problem was solved by maximum weight matching algorithm (i.e., assignment problem). These constraints assume the capacity of each edge as 1. But constraint 5 will ask for capacity of an edge  $\geq 1$ . (e.g. 3 optional karakas must be filled in a group of 5 optional karaka). With these constraints the problem no more remains an assignment problem, but reduces to min cost flow problem (which is a combination of both min cost problem and max flow problem).

### 3.3.1. Min cost flow Problem:

Let  $G=(V,A)$  be a directed network with a cost  $c_{ij}$  and a capacity  $u_{ij}$  associated with every arc  $(i,j)$  upon whether  $b(i) > 0$  or  $b(i) < 0$ . The minimum cost flow problem can be stated as follows.

$$\text{Minimize } z(x) = \sum_{(i,j) \in A} c_{ij} x_{ij}$$

subject to

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = b(i)$$

for all  $i \in N$

$$0 \leq x_{ij} \leq u_{ij} \quad \text{for all } (i,j) \in A.$$

### 3.3.2. Formation of G:

Formation of G is done in 2 stages.

1. Formation of V (set of nodes)
2. Formation of E (set of edges)

#### 3.3.2.1. Formation of VV:

There are 5 types of nodes

1. all source word groups  $S=(s_1, s_2, \dots, s_n)$
2. all demand karakas  $V=(v_1, v_2, \dots, v_m)$
3. source (SS)
4. tank (T)
5. intermediate nodes (E)

The set E is created as follows. If a group of optional karakas have the constraint 5, then a node  $N_1$  is created and edges are created from  $N_1$  to all nodes in that group of optional demand karakas.

### 3.3.2.2. Formation of E:

Each edge has information about cost of the edges, the lower bound of flow and the upper bound of flow

There are 5 types of edges.

1. Edges from source to all source word groups for which

Lower bound of flow = 1

Upper bound of flow = 1

cost of the edges = 0

2. Edges from source word groups to demand karakas. The edges are created by the formation of edges block.

Lower bound of flow = 0

Upper bound of flow = 1

cost of edges will be assigned by weight assignment block.

3. Edges from demand karakas to intermediate nodes

Lower bound of flow = 0

Upper bound of flow = 1

cost of edges = 0

4. Edges from intermediate nodes to tank

For these edges the lower bound of flow and upper bound of flow will be constraint specifies capacities.

(For example if 3 optional karakas out of 6 must be filled then

lower bound of edge = 3

upper bound of edge = 6)

cost of edges = 0

5. edges from demand karakas to tank:

lower bound of flow = 0 if optional karaka

1 if mandatory

upper bound of flow = 1

cost of edges = 0.

example 2 :

Let  $n_1, n_2, n_3, n_4, v_1, v_2$  be the given sentence

where  $n_1, n_2, n_3$  and  $n_4$  are source word groups.

Let the karaka charts of  $v_1$  and  $v_2$  be

$$v_1$$

1	m	
2	o	
3	o	

$$v_2$$

1	m	
2	o	

and the constraint is, one out of  $\{2,3\}$  of  $v_1$  must be filled. (in addition to constraints 1,2,3 and 4).

The set  $V$  is union of  $\{S, V, SS, T, E\}$

here  $S = \{n_1, n_2, n_3, n_4\}$

$V = \{v_{11}, v_{12}, v_{13}, v_{21}, v_{22}\}$

The constraint 5 is applicable to only one group, i.e.,  $v_{12}$  and  $v_{13}$ . So there is only one intermediate node. Let it be  $e_1$ ,

then  $V = \{n_1, n_2, n_3, n_4, v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, e_1, SS, T\}$

Let every element in  $S$  be a candidate to every demand karaka in  $V$  then the graph  $G = (V, A)$  is as shown in Fig 3.4.

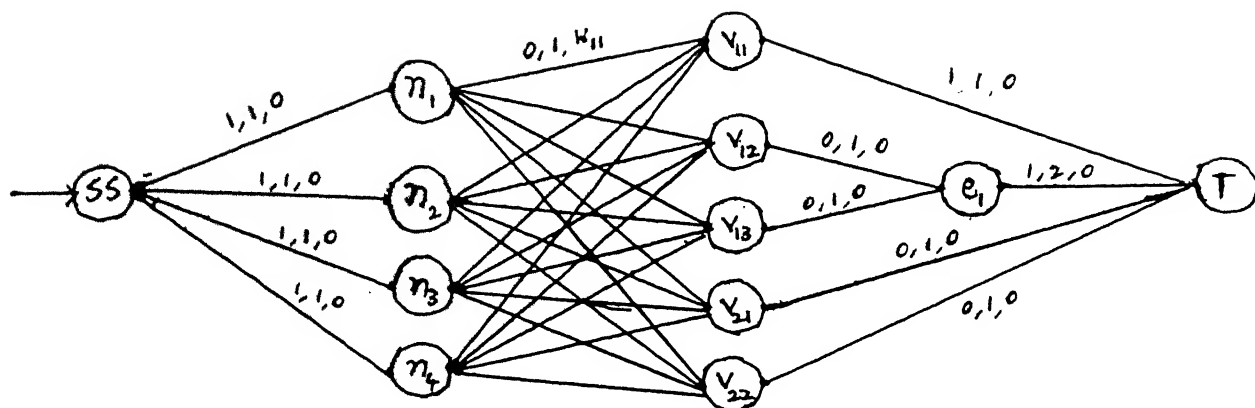


Fig 3.4. Graph  $G=(V,V,A)$  for example 2.

The graph  $G=(V,V,A)$  shown in Fig.3.4 was created and given as input to the min cost flow algorithm block [Ahuja] and the parsed structure is the output. The core parser flow chart is given in Fig 3.5.

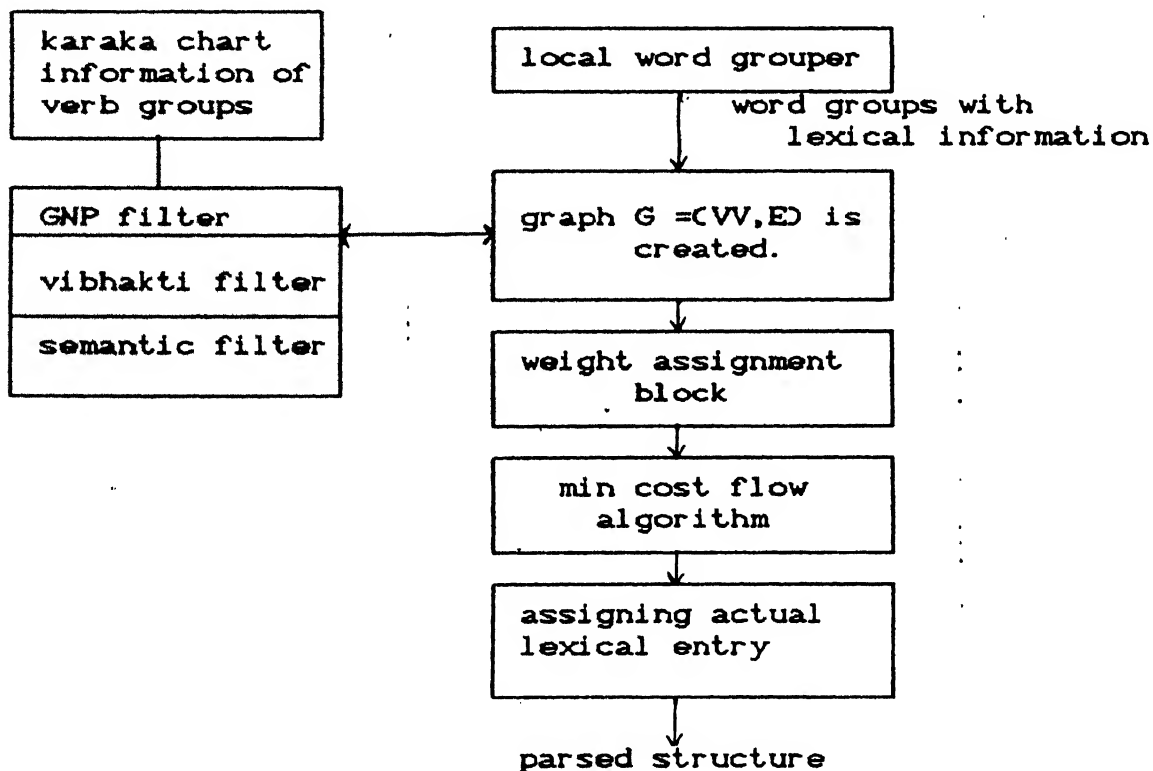


Fig 3.5. Flow chart of the core parser with constraints 1 to 5

In the formation of  $G$  the formation of intermediate nodes and edges containing the intermediate nodes may increase the complexity because if there are  $n$  optional karakas then a maximum of  $2^n - (n+1)$  intermediate nodes will be formed. But if the following assumption 1 is made then the complexity of the formation of graph  $G$  problem will still remain polynomial.

**Assumption 1 :** The intersection of groups is null.

With this assumption, the number of intermediate nodes will be linear to the number of karakas. So the formation of graph and the parser algorithm are polynomial.

For many natural language sentences the assumption holds (I didn't get any counter example).

Up to this stage the parsing problem (with constraints 1,2,3,4 and 5) was reduced to min cost flow problem whose complexity is polynomial time. So the parsing problem complexity with constraints 1,2,3,4 and 5 is polynomial. This can also be proved by total unimodularity and unimodularity conditions on the coefficient matrix  $A$  of the integer programming problem. The details of integer programming, Total unimodularity and unimodularity are discussed in section 3.4.

### 3.4. Integer Programming Approach:

**Integer programming problem:** Given a rational matrix  $A$  and rational vectors  $b$  and  $c$

determine  $\max \{ cx \mid \text{where } Ax \leq b, x \text{ integral} \}$

Total unimodularity: A matrix is said to be total unimodular if each sub determinant is +1, 0 or -1.

Theorem: All network matrices are total unimodular.

Unimodularity: A matrix is said to be unimodular if the determinant of every basis matrix is +1 or -1.

Let  $U$  be a non singular matrix.  $U$  is called unimodular if

$U$  is integral and has determinant  $\pm 1$

or every basis matrix of  $U$  has determinant  $\pm 1$ .

Theorem: Every total unimodular matrix is unimodular.

Each total unimodular matrix arises from either Network matrix or the matrices of the type

$$\begin{array}{cc} \begin{matrix} 1 & -1 & 0 & 0 & -1 \\ -1 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 1 \end{matrix} & \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{matrix} \end{array}$$

So if a matrix  $A$  is totally unimodular then the problem can be reduced to network problem if the matrix  $A$  has not arisen from other type of matrices specified above [Schri86].

Total unimodularity of a given matrix can be tested in polynomial time.

Therefore, a given matrix can be tested for being a network matrix in polynomial time.

If a matrix  $A$  is unimodular then for each integral vector  $b$ , the polyhedron

$$\{ x \mid x \geq 0 ; Ax=b \} \text{ is integral.}$$

In other words the problem can be solved by linear programming (still getting the integer solutions because the polyhedron vertices are all integers).

The above conditions (total unimodularity and unimodularity) are applied to the integer programming problem formulated from the parsing problem. The results are discussed in the following sections.

#### 3.4.1 With constraints 1 to 5:

With the constraints 1 to 5 the parsing problem was formulated as an integer programming problem (discussed in section 2.3.). The coefficient matrix of this formulation, if tested for total unimodularity, it will satisfy the test. Also if the matrix was tested for network matrix, it will satisfy that resulting that the parsing problem can be reduced to min cost flow problem.

#### 3.4.2 With constraints 1 to 6:

With constraints 1 to 6 the parsing problem will be formulated as an integer programming problem. The coefficient matrix A will be tested for total unimodularity and unimodularity conditions. If matrix A satisfies the properties then it can be stated that the parsing problem can be solved by using either min cost flow problem or linear programming problem whose time complexity is polynomial. So if the matrix satisfies the conditions time complexity of the parsing problem will be polynomial.

Now an example is given which will show that matrix A is not always unimodular.

Example 3 :        Let  $n_1, n_2, v_1, v_2$  be the given sentence  
                       where  $n_1, n_2$  are source word groups  
                       and  $v_1, v_2$  are demand word groups  
 Let the karaka charts of  $v_1$  and  $v_2$  be

$v_1$   

1	o	
2	o	
3	o	

$v_2$   

1	o	
---	---	--

Here  $v_1$  has 3 karaka roles and all are optional.  $v_2$  has 1 karaka role which is optional.

Let the graph be

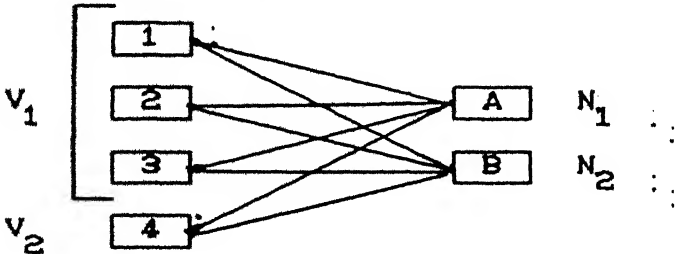


Fig 3.6. Graph of example 3.

Here each source word is a candidate for every karaka role.

The integer programming formulation with constraints 1 to 6 is

$$x_{1A} + x_{2A} + x_{3A} + x_{4A} = 1$$

$$x_{1B} + x_{2B} + x_{3B} + x_{4B} = 1$$

$$x_{1A} + x_{1B} \leq 1$$

$$x_{2A} + x_{2B} \leq 1$$

$$x_{3A} + x_{3B} \leq 1$$

$$x_{4A} + x_{4B} \leq 1$$

$$x_{1A} - x_{2B} - x_{3B} \leq 1$$

$$x_{2A} - x_{1B} - x_{3B} \leq 1$$

$$x_{3A} - x_{1B} - x_{2B} \leq 1$$

$$x_{ij} = 0 \text{ or } 1.$$

After adding the slack variables the coefficient matrix of this integer programming problem is

$x_{1A}$	$x_{2A}$	$x_{3A}$	$x_{4A}$	$x_{1B}$	$x_{2B}$	$x_{3B}$	$x_{4B}$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	-1	-1	0	0	0	0	0	1	0	0
0	1	0	0	-1	0	-1	0	0	0	0	0	0	1	0
0	0	1	0	-1	-1	0	0	0	0	0	0	0	0	1

After interchanging rows and columns the matrix was reduced to

$y_5$	$y_6$	$y_7$	$x_{1A}$	$x_{2A}$	$x_{3A}$	$x_{4A}$	$x_{4B}$	$y_1$	$y_2$	$y_3$	$y_4$	$x_{1B}$	$x_{2B}$	$x_{3B}$	
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	
0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	
0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	
0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	
0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	0	0	0	0	0	-1	-1	
0	1	0	0	1	0	0	0	0	0	0	0	-1	0	-1	
0	0	1	0	0	1	C		0	0	0	0	0	-1	-1	0

The basis matrix C is considered. After some transformations C will become

1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	-1	-1	0
0	0	0	0	0	0	-1	0	-1	0
0	0	0	0	0	0	-1	-1	0	0

whose determinant is -2.

So the matrix is not unimodular. It means that the sufficient condition was violated. After adding nesting constraint (constraint 6) it is not known whether parsing problem can be solved by linear programming. But with assumption 2 the problem can be again solved in polynomial time.

Assumption 2: Only mandatory karakas are present, i.e., there are no optional karakas.

Initially, for a karaka, all the source groups to the left of the verb group of that karaka are candidates. After that the karaka restrictions and constraints are applied. If a verb group has  $N$  karakas and all are mandatory then only  $N$  source groups to the left of that verb group will become candidates, in order to satisfy the nesting constraint.

Example 4: Let  $n_1 n_2 n_3 n_4 v_1 v_2$  be the given sentence. If the number of karakas of  $v_1$  are two and all are mandatory,  $n_2$  cannot satisfy any of the karakas of  $v_1$ , because the nesting constraint will be violated thus,  $n_2$  is not a candidate for any of the karakas of  $v_1$ .

Applying this domain restriction algorithm will satisfy the nesting constraint. The parsing problem without nesting constraint is solvable by min cost flow problem which has polynomial time complexity. So if only mandatory karakas are there then the parsing problem is solvable in polynomial time.

Observation-1 can be made about the nesting constraint.

Observation-1: Only optional karakas will take place in the nesting constraint (edges of optional karakas will form the nesting constraint equations).

Initially, for each karaka the source groups to the left of the verb group of that karaka are acceptable candidates. But if the verb group has M karakas ( both mandatory and optional) the (M+1)th source group to the left of verb group cannot become a candidate for the karakas, in order to satisfy the nesting constraint.

Example 5: Let  $n_1 n_2 n_3 n_4 n_5 n_6 v_1 v_2$  be the input sentence. Let  $v_1$  be having 1 mandatory karaka and 2 optional karakas then  $n_3$  cannot become a candidate to the karaka of  $v_1$ . So  $n_4 n_5 n_6$  are the acceptable candidates for karaka of  $v_1$ .

Now the set of elements which are acceptable to a verb group are found. Let it be P. Now a subset which satisfy the mandatory karaka requirement will be found from the above set. For each mandatory karaka, the vibhakti constraint will be applied to the source word groups, starting from the first element to the left of verb group moving towards left side. Let  $M_i$  be the set of source word group that satisfy the ith mandatory karaka (For example,  $M_1$  is the set of source word group that satisfies the first mandatory karaka, say karta). The union of all  $M_i$ 's will give the set which will satisfy all mandatory karaka requirement.

$$S = \bigcup_{i=1}^k M_i \quad (k \text{ mandatory karakas})$$

$$\text{and } S \subseteq P.$$

The elements of  $P-S$  will be acceptable candidates for optional karakas of this verb group or some other verb group followed by some other verb group.

So, for all mandatory karakas the set of source groups, which cannot become candidates for another verb group are formed. So the nesting constraint will not have any mandatory karakas. Only edges of optional karakas will form the nesting constraint equations.

Another assumption is considered.

Assumption 3: There are at most two edges in the graph with each optional karaka (i.e., at most two source groups can become candidates for an optional karaka).

With this assumption all the examples constructed turn out to have unimodular coefficient matrices of integer programming. An example is given below.

Example 6: Let  $n_1$   $n_2$   $n_3$   $v_1$   $v_2$  be the sentence.

Let the karaka charts of  $v_1$  and  $v_2$  are.

$v_1$			$v_2$		
1	o		1	o	
2	o		2	o	

Each source group can be a candidate for each karaka. The initial graph is given in Fig 3.7.

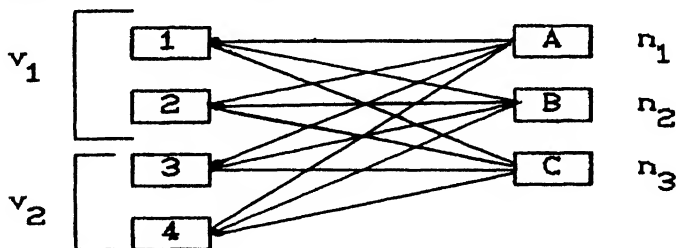


Fig 3.7. Graph of example 6

After the domain decision procedure the graph reduces to the graph in Fig 3.8.

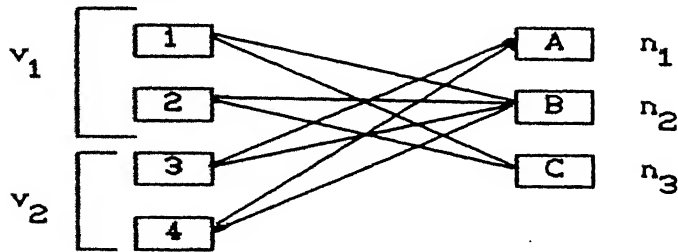


Fig 3.8. Reduced graph of example 6

Because  $n_1$  cannot become candidate to  $v_1$  and  $n_3$  cannot become candidate to  $v_2$  the integer programming formulation is

$$x_{3A} + x_{4A} = 1$$

$$x_{1B} + x_{2B} + x_{3B} + x_{4B} = 1$$

$$x_{1C} + x_{2C} = 1$$

$$x_{1C} + x_{1B} \leq 1$$

$$x_{2C} + x_{2B} \leq 1$$

$$x_{3A} + x_{3B} \leq 1$$

$$x_{4A} + x_{4B} \leq 1$$

$$x_{1B} - x_{2C} \leq 0$$

$$x_{2B} - x_{1C} \leq 0$$

$$x_{ij} = 0 \text{ or } 1.$$

and the coefficient matrix of this formulation is unimodular. Similar examples are formed and no counter example was found yet.

Assumption 3, made earlier, was satisfied by most of the natural language sentences that we checked.

With assumption 3 whether the coefficient matrix always satisfies the unimodular property is also an open problem.

### 3.5. Complexity Aspects:

1. With constraints 1,2 and 3 only the problem was reduced to bipartite matching whose time complexity is

$$O(\min(|U|, |V|) \cdot |E|)$$

2. With constraints 1 to 4 the problem was reduced to maximum bipartite matching whose time complexity is

$$O(n^3) \quad \text{where } |U|=|V|=n.$$

3. With constraints 1 to 5 the problem was reduced to min cost flow problem whose complexity is  $O((m \log n)(m + n \log n))$

where  $n$ =number of nodes,  $m$ = number of edges.

4. With all 6 constraints the problem was solved by integer programming.

The flow chart of the core parser is given in Fig 3.9.

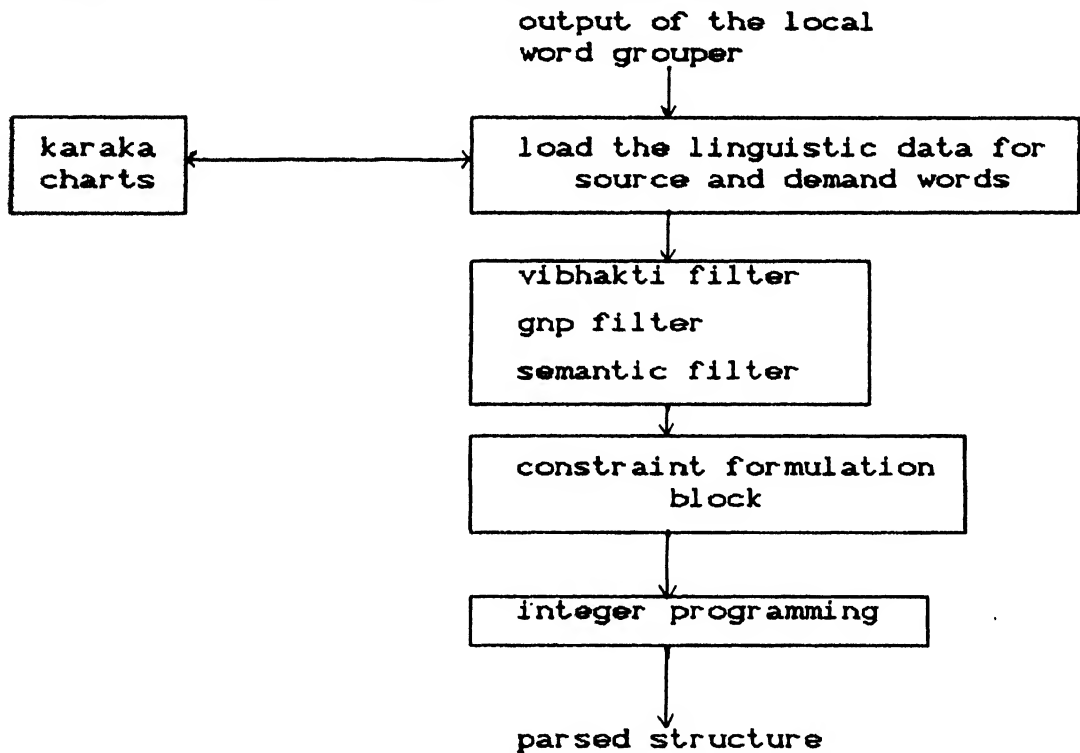


Fig 3.9. Flow chart of the core parser.

The above specified core parser complexity aspects are valid only if the following assumptions are true.

**Assumptions 4:**

1. There are no source word, demand word clashes  
(in other words there are no category clashes).
2. The merged karaka charts can be formed for all senses of verb group.

The assumptions can be relaxed in a controlled fashion.

**3.5.1. Increasing number of merged karaka charts:**

Now the assumption is that only one merged karaka chart can be formed. By this the core parser flow chart will be executed only once because there will be a unique demand group. If more than one karaka charts are there for each verb group, the total number of times the core parser flowchart executed will increase.

If  $k_1, k_2, k_3, \dots, k_n$  are the number of karaka charts for each verb  $v_1, v_2, \dots, v_n$  then the core parser flow chart will be executed  $n^k$  times.

If at most a fixed 'k' number of merged karaka charts are formed for each verb group then the complexity of problem increases but remains to some constant times that of core parser complexity because k is bounded.

### 3.5.2 Allowing source word, demand word clashes:

If all words in the sentence can be source words and demand words there will be  $2^n$  of demand word selections, where  $n$  is the number of words. So the core parser algorithm has to be executed exponential times.

However, a study of the lexical data gives the information that 1% of the words in the given sentence have clashes. This information is based on the analysis of a kannada corpus of a few hundred thousand words.

## CONCLUSIONS

The machine translation system consists of a morphological analyzer, a local word grouper and a core parser. The complexity of the system is dependant on these three blocks. The time complexity of the first two blocks is polynomial. But the complexity of the core parser is dependant on three aspects. They are

1. Number of category clashes
2. Number of karaka charts associated with each verb group.
3. Constraints.

The first two aspects are mainly lexicon dependant. But the constraints play an important role in the complexity of the core parser. With some of the constraints the problem will remain polynomial (time) (as in the case with constraints 1 to 5). But with the inclusion of the island constraint (6 th constraint) the complexity of the problem is not known exactly. The problem was reduced to the integer programming problem.

The parser should give unique parse structure. Otherwise more than one translations will be produced for a given sentence. The reason for the ambiguous parse structures is the idiosyncrasies in the language like word overloading of words

which give category clashes.

Extensions can be done to the present work in the following directions.

1. constraints: Presently there are 6 constraints in the system. It has to be studied whether more constraints are required to get a unique parse structure, if so what is the complexity of the system

2. With the Island constraint the parsing problem was reduced to integer programming. But the parsing problem complexity with the island constraint is not known. It has to be studied whether the problem is polynomial time complex or NP-complete or NP hard.

## REFERENCES

[Ahuja]

Ahuja. R.K., Thomas L. Magnanti and James B. Orlin, Network flows; Theory, Algorithms and applications, to be published by Prentice-hall, Inc.

[Bhanu89]

Bhanumati. B., An approach to Machine Translation among Indian languages, Technical Report TRCS-89-90, Department of Computer Science and Engineering, IIT, Kanpur, 1989.

[Bhar89]

Bharati Akshar, Vineet Chaitanya and Rajeev Sangal, A karaka based approach to parsing of Indian languages, Technical report TRCS-89-88, Department of Computer Science and Engg., IIT, Kanpur, 1989.

[Bhar90]

Bharati Akshar, Vineet Chaitanya and Rajeev Sangal, Computational grammer for Indian languages processing, Technical report TRCS-90-91, Department of Computer Science and Engg., IIT, Kanpur, 1990.

[Krish85]

Krishnamurty B.H. and J.P.L. Gwynn, A grammer of Modern Telugu, Oxford University Press, Delhi, 1985.

[Niren87]

Nirenberg Sergei (ed.), Machine Translation: Theoretical and Methodological issues, (studies in Natural language processing), Cambridge University Press, Cambridge, 1987.

[Papa82]

Papadimitriou, H. Christos and Kenneth Steiglitz,  
Combinatorial Optimization: Algorithms and complexity,  
Prentice-hall,1982.

[Ravi92]

Ravishankar P.V., Enhancements to the Linguist's workbench,  
M.tech Thesis, Department of Computer Science and Engg., IIT,  
Kanpur, 1992.

[Salk78]

Salkin Harvey M., Integer Programming, Addison Wesley,1975.

[Schri86]

Schrijver Alexander, Theory of linear and integer  
programming., Chichester wiley,1986.

[Srin91]

Srinivas B., Linguist's workbench - A grammar development  
tool for Indian languages, M.tech Thesis, Department of  
Computer Science and Engg., IIT, Kanpur, 1991.

# Appendix

## Devanagiri-Roman alphabet mapping

अ	आ	इ	ई	उ	ऊ	ऋ	ॠ	ए	ऐ	ओ	औ	ह	ः	ॐ
				k	K	g	G	f						
				क	ख	ग	घ	ङ						
				c	C	j	J	F						
				च	छ	ज	झ	ञ						
				t	T	d	D	N						
				ट	ठ	ड	ढ	ण						
				w	W	∞	X	n						
				व	य	ॠ	ष	न						
				p	P	b	B	m						
				प	फ	ब	भ	म						
				y	r	l	v							
				य	र	ल	व							
				S	R	s	h							
				श	ष	स	ह							

Examples: rAma राम kqRNa कृष्ण jFAana जान Sawru शत्रु AzKa अख yakRa यम